**CODING ASSIGNMENT – 2 ( PYTHON)**

**BY AMITA C**

**### \*\*Exercise 1: Creating DataFrame from Scratch\*\***

**1. Create a DataFrame with the following columns: `"Product"`, `"Category"`, `"Price"`, and `"Quantity"`. Use the following data:**

**- Product: `['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone']`**

**- Category: `['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics']`**

**- Price: `[80000, 1500, 20000, 3000, 40000]`**

**- Quantity: `[10, 100, 50, 75, 30]`**

import pandas as pd

data = {

'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'],

'Category': ['Electronics', 'Accessories', 'Electronics', 'Accessories', 'Electronics'],

'Price': [80000, 1500, 20000, 3000, 40000],

'Quantity': [10, 100, 50, 75, 30]

}

df = pd.DataFrame(data)

**2. Print the DataFrame.**

print("DataFrame:\n", df)

**### \*\*Exercise 2: Basic DataFrame Operations\*\***

1. **Display the first 3 rows of the DataFrame.**

print("First 3 rows:\n", df.head(3))

1. **Display the column names and index of the DataFrame.**

print("\nColumn Names:", df.columns) print("Index:", df.index)

1. **Display a summary of statistics (mean, min, max, etc.) for the numeric columns in the DataFrame.**

print("\nSummary statistics:\n", df.describe())

**### \*\*Exercise 3: Selecting Data\*\***

1. **Select and display the `"Product"` and `"Price"` columns.**

print("Product and Price columns:\n", df[['Product', 'Price']])

1. **Select rows where the `"Category"` is `"Electronics"` and print them.**

electronics\_df = df[df['Category'] == 'Electronics'] print("\nRows where Category is 'Electronics':\n", electronics\_df)

**### \*\*Exercise 4: Filtering Data\*\***

1. **Filter the DataFrame to display only the products with a price greater than `10,000`.**

price\_filter = df[df['Price'] > 10000] print("Products with Price > 10,000:\n", price\_filter)

1. **Filter the DataFrame to show only products that belong to the `"Accessories"` category and have a quantity greater than `50`.**

accessories\_filter = df[(df['Category'] == 'Accessories') & (df['Quantity'] > 50)] print("\n'Accessories' category with Quantity > 50:\n", accessories\_filter)

**### \*\*Exercise 5: Adding and Removing Columns\*\***

1. **Add a new column `"Total Value"` which is calculated by multiplying `"Price"` and `"Quantity"`.**

df['Total Value'] = df['Price'] \* df['Quantity']

print("DataFrame with 'Total Value':\n", df)

1. **Drop the `"Category"` column from the DataFrame and print the updated DataFrame.**

df\_dropped = df.drop('Category', axis=1)

print("\nDataFrame without 'Category' column:\n", df\_dropped)

**### \*\*Exercise 6: Sorting Data\*\***

1. **Sort the DataFrame by `"Price"` in descending order.**

df\_sorted\_price = df.sort\_values('Price', ascending=False)

print("DataFrame sorted by 'Price' (desc):\n", df\_sorted\_price)

1. **Sort the DataFrame by `"Quantity"` in ascending order, then by `"Price"` in descending order (multi-level sorting).**

df\_sorted\_multi = df.sort\_values(['Quantity', 'Price'], ascending=[True, False]) print("\nDataFrame sorted by 'Quantity' (asc) then 'Price' (desc):\n", df\_sorted\_multi)

**### \*\*Exercise 7: Grouping Data\*\***

1. **Group the DataFrame by `"Category"` and calculate the total quantity for each category.**

grouped\_quantity = df.groupby('Category')['Quantity'].sum()

print("Total Quantity by Category:\n", grouped\_quantity)

1. **Group by `"Category"` and calculate the average price for each category.**

grouped\_price = df.groupby('Category')['Price'].mean()

print("\nAverage Price by Category:\n", grouped\_price)

**### \*\*Exercise 8: Handling Missing Data\*\***

1. **Introduce some missing values in the `"Price"` column by assigning `None` to two rows.**

df\_missing = df.copy()

df\_missing.loc[1, 'Price'] = None

df\_missing.loc[3, 'Price'] = None

print("DataFrame with missing 'Price' values:\n", df\_missing)

1. **Fill the missing values with the mean price of the available products.**

df\_missing['Price'].fillna(df\_missing['Price'].mean(), inplace=True)

print("\nDataFrame after filling missing 'Price' values:\n", df\_missing)

1. **Drop any rows where the `"Quantity"` is less than `50`.**

df\_filtered = df[df['Quantity'] >= 50]

print("\nDataFrame after dropping rows with Quantity < 50:\n", df\_filtered)

**### \*\*Exercise 9: Apply Custom Functions\*\***

1. **Apply a custom function to the `"Price"` column that increases all prices by 5%.**

df['Price'] = df['Price'].apply(lambda x: x \* 1.05)

print("DataFrame after increasing 'Price' by 5%:\n", df)

1. **Create a new column `"Discounted Price"` that reduces the original price by 10%.**

df['Discounted Price'] = df['Price'] \* 0.90

print("\nDataFrame with 'Discounted Price':\n", df)

**### \*\*Exercise 10: Merging DataFrames\*\***

1. **Create another DataFrame with columns `"Product"` and `"Supplier"`, and merge it with the original DataFrame based on the `"Product"` column.**

supplier\_data = { 'Product': ['Laptop', 'Mouse', 'Monitor', 'Keyboard', 'Phone'], 'Supplier': ['Dell', 'Logitech', 'Samsung', 'HP', 'Apple'] }

df\_supplier = pd.DataFrame(supplier\_data)

df\_merged = pd.merge(df, df\_supplier, on='Product')

print("Merged DataFrame:\n", df\_merged)

**### \*\*Exercise 11: Pivot Tables\*\***

1. **Create a pivot table that shows the total quantity of products for each category and product combination.**

pivot\_table = df.pivot\_table(values='Quantity', index='Category', columns='Product', aggfunc='sum')

print("Pivot Table (Total Quantity by Category and Product):\n", pivot\_table)

**### \*\*Exercise 12: Concatenating DataFrames\*\***

1. **Create two separate DataFrames for two different stores with the same columns (`"Product"`, `"Price"`, `"Quantity"`).**

data\_store\_1 = {

'Product': ['Laptop', 'Mouse', 'Monitor'],

'Price': [80000, 1500, 20000],

'Quantity': [5, 50, 20]

}

data\_store\_2 = {

'Product': ['Keyboard', 'Phone', 'Monitor'],

'Price': [3000, 40000, 20000],

'Quantity': [30, 15, 10]

}

df\_store\_1 = pd.DataFrame(data\_store\_1)

df\_store\_2 = pd.DataFrame(data\_store\_2)

1. **Concatenate these DataFrames to create a combined inventory list.**

combined\_inventory = pd.concat([df\_store\_1, df\_store\_2], ignore\_index=True)

print("Combined Inventory List:\n", combined\_inventory)

**### \*\*Exercise 13: Working with Dates\*\***

1. **Create a DataFrame with a `"Date"` column that contains the last 5 days starting from today.**

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

dates = [datetime.today() - timedelta(days=i) for i in range(5)]

sales\_data = np.random.randint(1000, 5000, size=5)

df\_dates = pd.DataFrame({

'Date': dates,

'Sales': sales\_data

})

print("DataFrame with Dates and Sales:\n", df\_dates)

**2. Add a column `"Sales"` with random values for each day.**

**3. Find the total sales for all days combined.**

# Find the total sales for all days combined

total\_sales = df\_dates['Sales'].sum()

print("\nTotal Sales for all days combined:", total\_sales)

**### \*\*Exercise 14: Reshaping Data with Melt\*\***

1. **Create a DataFrame with columns `"Product"`, `"Region"`, `"Q1\_Sales"`, `"Q2\_Sales"`.**

data\_sales = {

'Product': ['Laptop', 'Mouse', 'Monitor'],

'Region': ['North', 'South', 'East'],

'Q1\_Sales': [15000, 12000, 8000],

'Q2\_Sales': [20000, 11000, 9000]

}

df\_sales = pd.DataFrame(data\_sales)

1. **Use `pd.melt()` to reshape the DataFrame so that it has columns `"Product"`, `"Region"`, `"Quarter"`, and `"Sales"`.**

melted\_df = pd.melt(df\_sales, id\_vars=['Product', 'Region'], var\_name='Quarter', value\_name='Sales')

print("Melted DataFrame:\n", melted\_df)

**### \*\*Exercise 15: Reading and Writing Data\*\***

1. **Read the data from a CSV file named `products.csv` into a DataFrame.**

df\_products = pd.read\_csv('products.csv')

print("DataFrame from CSV:\n", df\_products)

1. **After performing some operations (e.g., adding a new column or modifying values), write the DataFrame back to a new CSV file named `updated\_products.csv`.**

df\_products['Discount'] = df\_products['Price'] \* 0.10

df\_products.to\_csv('updated\_products.csv', index=False)

print("\nUpdated DataFrame written to 'updated\_products.csv'")

**### \*\*Exercise 16: Renaming Columns\*\***

1. **Given a DataFrame with columns `"Prod"`, `"Cat"`, `"Price"`, `"Qty"`, rename the columns to `"Product"`, `"Category"`, `"Price"`, and `"Quantity"`.**

import pandas as pd

df = pd.DataFrame({

'Prod': ['Laptop', 'Mouse', 'Keyboard'],

'Cat': ['Electronics', 'Electronics', 'Electronics'],

'Price': [80000, 1500, 3000],

'Qty': [5, 50, 30]

})

1. **Print the renamed DataFrame.**

df.rename(columns={'Prod': 'Product', 'Cat': 'Category', 'Qty': 'Quantity'}, inplace=True)

print("Renamed DataFrame:\n", df)

**### \*\*Exercise 17: Creating a MultiIndex DataFrame\*\***

1. **Create a DataFrame using a MultiIndex (hierarchical index) with two levels: `"Store"` and `"Product"`. The DataFrame should have columns `"Price"` and `"Quantity"`, representing the price and quantity of products in different stores.**

arrays = [

['Store A', 'Store A', 'Store B', 'Store B'],

['Laptop', 'Mouse', 'Keyboard', 'Monitor']

]

index = pd.MultiIndex.from\_arrays(arrays, names=('Store', 'Product'))

df\_multiindex = pd.DataFrame({

'Price': [80000, 1500, 3000, 20000],

'Quantity': [10, 50, 20, 15]

}, index=index)

1. **Print the MultiIndex DataFrame.**

print("MultiIndex DataFrame:\n", df\_multiindex)

**### \*\*Exercise 18: Resample Time-Series Data\*\***

1. **Create a DataFrame with a `"Date"` column containing a range of dates for the past 30 days and a `"Sales"` column with random values.**

import numpy as np

from datetime import datetime, timedelta

date\_range = pd.date\_range(end=datetime.today(), periods=30)

sales\_data = np.random.randint(1000, 5000, size=30)

df\_time\_series = pd.DataFrame({

'Date': date\_range,

'Sales': sales\_data

})

print("Time-Series DataFrame:\n", df\_time\_series)

1. **Resample the data to show the total sales by week.**

df\_resampled = df\_time\_series.set\_index('Date').resample('W').sum()

print("\nTotal Sales by Week:\n", df\_resampled)

**### \*\*Exercise 19: Handling Duplicates\*\***

1. **Given a DataFrame with duplicate rows, identify and remove the duplicate rows.**

df\_duplicates = pd.DataFrame({

'Product': ['Laptop', 'Mouse', 'Laptop', 'Keyboard'],

'Price': [80000, 1500, 80000, 3000],

'Quantity': [5, 50, 5, 20]

})

print("Original DataFrame:\n", df\_duplicates)

duplicates = df\_duplicates[df\_duplicates.duplicated()]

print("\nDuplicate Rows:\n", duplicates)

1. **Print the cleaned DataFrame.**

df\_cleaned = df\_duplicates.drop\_duplicates()

print("\nCleaned DataFrame:\n", df\_cleaned)

**### \*\*Exercise 20: Correlation Matrix\*\***

1. **Create a DataFrame with numeric data representing different features (e.g., `"Height"`, `"Weight"`, `"Age"`, `"Income"`).**

df\_numeric = pd.DataFrame({

'Height': [170, 160, 175, 180, 165],

'Weight': [65, 70, 72, 85, 60],

'Age': [25, 30, 35, 40, 22],

'Income': [50000, 60000, 70000, 80000, 55000]

})

1. **Compute the correlation matrix for the DataFrame.**

correlation\_matrix = df\_numeric.corr()

1. **Print the correlation matrix.**

print("Correlation Matrix:\n", correlation\_matrix)

**### \*\*Exercise 21: Cumulative Sum and Rolling Windows\*\***

1. **Create a DataFrame with random sales data for each day over the last 30 days.**

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

date\_range = pd.date\_range(end=datetime.today(), periods=30)

sales\_data = np.random.randint(1000, 5000, size=30)

df\_sales = pd.DataFrame({

'Date': date\_range,

'Sales': sales\_data

})

print("Sales DataFrame with Cumulative Sum and Rolling Avg:\n", df\_sales)

1. **Calculate the cumulative sum of the sales and add it as a new column `"Cumulative Sales"`.**

df\_sales['Cumulative Sales'] = df\_sales['Sales'].cumsum()

1. **Calculate the rolling average of sales over the past 7 days and add it as a new column `"Rolling Avg"`.**

df\_sales['Rolling Avg'] = df\_sales['Sales'].rolling(window=7).mean()

print("Sales DataFrame with Cumulative Sum and Rolling Avg:\n", df\_sales)

**### \*\*Exercise 22: String Operations\*\***

1. **Create a DataFrame with a column `"Names"` containing values like `"John Doe"`, `"Jane Smith"`, `"Sam Brown"`.**

df\_names = pd.DataFrame({

'Names': ['John Doe', 'Jane Smith', 'Sam Brown']

})

1. **Split the `"Names"` column into two separate columns: `"First Name"` and `"Last Name"`.**

df\_names[['First Name', 'Last Name']] = df\_names['Names'].str.split(' ', expand=True)

1. **Convert the `"First Name"` column to uppercase.**

df\_names['First Name'] = df\_names['First Name'].str.upper()

print("DataFrame with Split and Uppercase Names:\n", df\_names)

**### \*\*Exercise 23: Conditional Selections with `np.where`\*\***

1. **Create a DataFrame with columns `"Employee"`, `"Age"`, and `"Department"`.**

df\_employees = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Sara'],

'Age': [45, 34, 50, 28],

'Department': ['HR', 'IT', 'Finance', 'Marketing'

] })

1. **Create a new column `"Status"` that assigns `"Senior"` to employees aged 40 or above and `"Junior"` to employees below 40 using `np.where()`.**

df\_employees['Status'] = np.where(df\_employees['Age'] >= 40, 'Senior', 'Junior') print("DataFrame with Status based on Age:\n", df\_employees)

**### \*\*Exercise 24: Slicing DataFrames\*\***

**1. Given a DataFrame with data on `"Products"`, `"Category"`, `"Sales"`, and `"Profit"`, slice the DataFrame to display:**

**- The first 10 rows.**

**- All rows where the `"Category"` is `"Electronics"`.**

**- Only the `"Sales"` and `"Profit"` columns for products with sales greater than 50,000.**

df\_products = pd.DataFrame({

'Products': ['Laptop', 'Mouse', 'Keyboard', 'Monitor', 'Printer', 'Camera', 'Smartphone', 'Tablet', 'Router', 'Headset'],

'Category': ['Electronics', 'Electronics', 'Electronics', 'Electronics', 'Office', 'Electronics', 'Electronics', 'Electronics', 'Electronics', 'Electronics'],

'Sales': [75000, 2000, 4500, 50000, 12000, 25000, 80000, 35000, 10000, 7000],

'Profit': [15000, 500, 800, 20000, 3000, 5000, 25000, 7000, 1500, 1000]

})

first\_10\_rows = df\_products.head(10)

electronics\_category = df\_products[df\_products['Category'] == 'Electronics']

sales\_profit\_over\_50k = df\_products[df\_products['Sales'] > 50000][['Sales', 'Profit']]

print("First 10 Rows:\n", first\_10\_rows)

print("\nElectronics Category:\n", electronics\_category)

print("\nSales and Profit for Products with Sales > 50,000:\n", sales\_profit\_over\_50k)

**### \*\*Exercise 25: Concatenating DataFrames Vertically and Horizontally\*\***

1. **Create two DataFrames with identical columns `"Employee"`, `"Age"`, `"Salary"`, but different rows (e.g., one for employees in `"Store A"` and one for employees in `"Store B"`).**

df\_store\_a = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam'],

'Age': [28, 34, 40],

'Salary': [50000, 60000, 55000]

})

df\_store\_b = pd.DataFrame({

'Employee': ['Sara', 'Tom', 'Anna'],

'Age': [25, 45, 30],

'Salary': [52000, 62000, 58000] })

1. **Concatenate the DataFrames vertically to create a combined DataFrame.**

df\_combined\_vertical = pd.concat([df\_store\_a, df\_store\_b], ignore\_index=True)

print("Combined DataFrame (Vertical):\n", df\_combined\_vertical)

1. **Now create two DataFrames with different columns (e.g., `"Employee"`, `"Department"` and `"Employee"`, `"Salary"`) and concatenate them horizontally based on the common `"Employee"` column.**

df\_department = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Sara', 'Tom', 'Anna'],

'Department': ['HR', 'IT', 'Finance', 'Marketing', 'Sales', 'Support']

})

df\_salary = pd.DataFrame({

'Employee': ['John', 'Jane', 'Sam', 'Sara', 'Tom', 'Anna'],

'Salary': [50000, 60000, 55000, 52000, 62000, 58000] })

df\_combined\_horizontal = pd.merge(df\_department, df\_salary, on='Employee') print("\nCombined DataFrame (Horizontal):\n", df\_combined\_horizontal)

**### \*\*Exercise 26: Exploding Lists in DataFrame Columns\*\***

1. **Create a DataFrame with a column `"Product"` and a column `"Features"` where each feature is a list (e.g., `["Feature1", "Feature2"]`).**

import pandas as pd

df\_features = pd.DataFrame({

'Product': ['Laptop', 'Smartphone', 'Tablet'],

'Features': [['Touchscreen', 'Bluetooth'], ['Camera', '5G', 'Bluetooth'], ['WiFi', 'Touchscreen']]

})

1. **Use the `explode()` method to create a new row for each feature in the list, so each product-feature pair has its own row.**

df\_exploded = df\_features.explode('Features')

print("DataFrame after exploding 'Features':\n", df\_exploded)

**### \*\*Exercise 27: Using `.map()` and `.applymap()`\*\***

1. **Given a DataFrame with columns `"Product"`, `"Price"`, and `"Quantity"`, use `.map()` to apply a custom function to increase `"Price"` by 10% for each row.**

df\_products = pd.DataFrame({

'Product': ['Laptop', 'Smartphone', 'Tablet'],

'Price': [1000, 800, 500],

'Quantity': [10, 15, 20]

})

df\_products['Price'] = df\_products['Price'].map(lambda x: x \* 1.1)

1. **Use `.applymap()` to format the numeric values in the DataFrame to two decimal places.**

df\_products = df\_products.applymap(lambda x: f'{x:.2f}' if isinstance(x, (int, float)) else x)

print("DataFrame after applying map and applymap:\n", df\_products)

**### \*\*Exercise 28: Combining `groupby()` with `apply()`\*\***

1. **Create a DataFrame with `"City"`, `"Product"`, `"Sales"`, and `"Profit"`.**

df\_city\_sales = pd.DataFrame({

'City': ['New York', 'Los Angeles', 'New York', 'Los Angeles', 'Chicago'],

'Product': ['Laptop', 'Smartphone', 'Tablet', 'Laptop', 'Smartphone'],

'Sales': [150000, 200000, 100000, 120000, 130000],

'Profit': [30000, 40000, 20000, 24000, 26000]

})

def calculate\_margin(group):

return group['Profit'].sum() / group['Sales'].sum()

1. **Group by `"City"` and apply a custom function to calculate the profit margin (Profit/Sales) for each city.**

df\_profit\_margin = df\_city\_sales.groupby('City').apply(calculate\_margin).reset\_index(name='Profit Margin')

print("DataFrame with Profit Margin by City:\n", df\_profit\_margin)

**### \*\*Exercise 29: Creating a DataFrame from Multiple Sources\*\***

1. **Create three different DataFrames from different sources (e.g., CSV, JSON, and a Python dictionary).**

import pandas as pd

import json

data\_csv = {

'ID': [1, 2, 3],

'Name': ['John', 'Jane', 'Tom'],

'Age': [28, 34, 45]

}

df\_csv = pd.DataFrame(data\_csv)

data\_json = json.loads('''[

{"ID": 1, "Department": "HR"},

{"ID": 2, "Department": "IT"},

{"ID": 3, "Department": "Finance"}

]''')

df\_json = pd.DataFrame(data\_json)

data\_dict = {

'ID': [1, 2, 3],

'Salary': [50000, 60000, 70000]

}

df\_dict = pd.DataFrame(data\_dict)

1. **Merge the DataFrames based on a common column and create a consolidated report.**

df\_merged = pd.merge(pd.merge(df\_csv, df\_json, on='ID'), df\_dict, on='ID')

print("Consolidated DataFrame:\n", df\_merged)

**### \*\*Exercise 30: Dealing with Large Datasets\*\***

1. **Create a large DataFrame with 1 million rows, representing data on `"Transaction ID"`, `"Customer"`, `"Product"`, `"Amount"`, and `"Date"`.**

import pandas as pd

import numpy as np

n = 1\_000\_000

df\_large = pd.DataFrame({

'Transaction ID': np.arange(1, n + 1),

'Customer': np.random.choice(['Customer1', 'Customer2', 'Customer3', 'Customer4'], size=n),

'Product': np.random.choice(['ProductA', 'ProductB', 'ProductC'], size=n),

'Amount': np.random.uniform(10, 1000, size=n),

'Date': pd.date\_range(start='2023-01-01', periods=n, freq='T')

})

1. **Split the DataFrame into smaller chunks (e.g., 100,000 rows each), perform a simple analysis on each chunk (e.g., total sales), and combine the results.**

chunk\_size = 100\_000

chunks = [df\_large.iloc[i:i + chunk\_size] for i in range(0, len(df\_large), chunk\_size)]

# Perform analysis (e.g., total sales) on each chunk and combine the results

total\_sales = sum(chunk['Amount'].sum() for chunk in chunks)

print("Total Sales from all chunks: ", total\_sales)

**---**